**Deep Learning Homework I Report**

Name:Dazhi Li

RUID:197007456

**Problem 1**

Source Code:

import numpy as np

import heapq

from collections import Counter

#Input the training\_data

class\_A=[(0,1,0),(0,1,1),(1,2,1),(1,2,0)]

class\_B=[(1,2,2),(2,2,2),(1,2,-1),(2,2,3)]

class\_C=[(-1,-1,-1),(0,-1,-2),(0,-1,1),(-1,-2,1)]

sample=(1,0,1)

L2=[]

#Calculate the distance and remember the training\_data

for i in class\_A:

L2\_dist=np.sqrt((i[0]-sample[0])\*\*2+(i[1]-sample[1])\*\*2+(i[2]-sample[2])\*\*2)

new={'class':'class\_A','L2\_dist':L2\_dist}

L2.append(new)

for i in class\_B:

L2\_dist=np.sqrt((i[0]-sample[0])\*\*2+(i[1]-sample[1])\*\*2+(i[2]-sample[2])\*\*2)

new={'class':'class\_B','L2\_dist':L2\_dist}

L2.append(new)

for i in class\_C:

L2\_dist=np.sqrt((i[0]-sample[0])\*\*2+(i[1]-sample[1])\*\*2+(i[2]-sample[2])\*\*2)

new={'class':'class\_C','L2\_dist':L2\_dist}

L2.append(new)

#print(L2)

#Define the KNN classifier to get the output label

def KNN\_classifier(k):

if k==1:

print("\nWhen K=1,")

shortest=heapq.nsmallest(k+1, L2, key=lambda s: s['L2\_dist'])

if shortest[0]['L2\_dist']==shortest[1]['L2\_dist']:

print("Sample point "+str(sample)+" can be classified to :"+\

shortest[0]['class']+"/"+shortest[1]['class'])

else:

print("Sample point "+str(sample)+" can be classified to: "+\

shortest[0]['class'])

elif k==3:

print("\nWhen K=3,")

shortest=heapq.nsmallest(k, L2, key=lambda s: s['L2\_dist'])

A=[]

A.append(shortest[0]['class'])

A.append(shortest[1]['class'])

A.append(shortest[2]['class'])

B=Counter(A).most\_common(1)

if B[0][1]==1:

print("Sample point "+str(sample)+" can not be classified")

else:

print("Sample point "+str(sample)+" can be classified to :"+\

B[0][0])

elif k==5:

print("\nWhen K=5,")

shortest=heapq.nsmallest(k, L2, key=lambda s: s['L2\_dist'])

A=[]

A.append(shortest[0]['class'])

A.append(shortest[1]['class'])

A.append(shortest[2]['class'])

A.append(shortest[3]['class'])

A.append(shortest[4]['class'])

B=Counter(A).most\_common(1)

if B[0][1]==1:

print("Sample point "+str(sample)+" can not be classified")

else:

print("Sample point "+str(sample)+" can be classified to :"+\

B[0][0])

KNN\_classifier(1)

KNN\_classifier(3)

KNN\_classifier(5)

Program Procedure:

At first I input those training data and sample point into my program. Then I calculate the L2 distance between the sample point and all the training data. I use a list-dictionary structure named L2 to store all the L2 distances and their classifications. At first I just printed all the L2 data and found that if K=1, there would be two smallest L2 distances belongs to two different class. So I add some logical judge in KNN\_classifier method. There would be some special situations that there would be a point belongs to no one class as what we have seen the “white part” in our slides. Finally, I use heapq library to get the N smallest distances from the L2 structure. And I count how often a label of a class will occur in it as my final KNN output. The program if not so perfect that every kind of datasets would work. But it is still enough to work in most of the situations. Finally, the output screen will be shown below in **figure 1**.

Output of Problem 1:

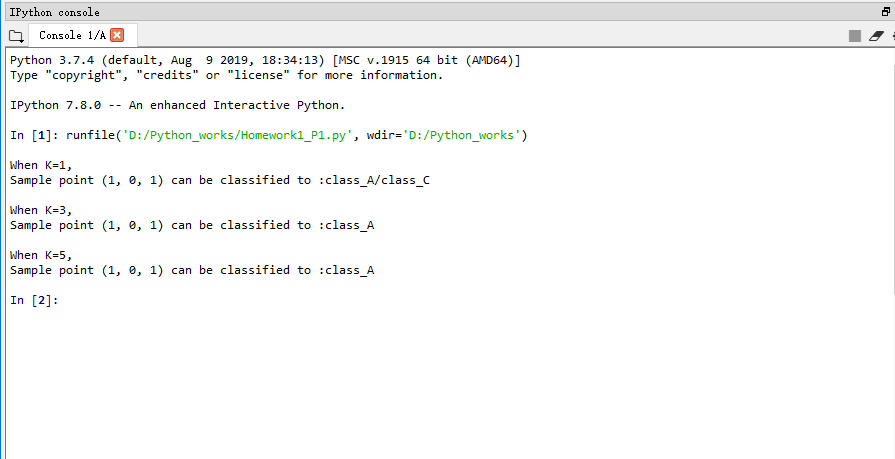


Figure 1

**Problem 2**

Source Code:

import numpy as np

import matplotlib as mpl

mpl.use('Agg')

import matplotlib.pyplot as plt

from collections import Counter

# load mini training data and labels

mini\_train = np.load('knn\_minitrain.npy')

mini\_train\_label = np.load('knn\_minitrain\_label.npy')

# randomly generate test data

mini\_test = np.random.randint(20, size=20)

mini\_test = mini\_test.reshape(10,2)

# Define knn classifier

def kNNClassify(newInput, dataSet, labels, k):

Inf=999

result=[]

########################

# Input your code here #

########################

for i in mini\_test:

L2=[]

for j in mini\_train:

L2\_dist=np.sqrt((i[0]-j[0])\*\*2+(i[1]-j[1])\*\*2)

L2.append(L2\_dist)

Min\_list=[]

for a in range(k):

Min\_list.append(L2.index(min(L2)))

L2[L2.index(min(L2))]=Inf

classifier=[]

for b in Min\_list:

classifier.append(mini\_train\_label[b])

#print(classifier)

result.append(Counter(classifier).most\_common(1)[0][0])

####################

# End of your code #

####################

return result

outputlabels=kNNClassify(mini\_test,mini\_train,mini\_train\_label,6)

print ('random test points are:', mini\_test)

print ('knn classfied labels for test:', outputlabels)

# plot train data and classfied test data

train\_x = mini\_train[:,0]

train\_y = mini\_train[:,1]

fig = plt.figure()

plt.scatter(train\_x[np.where(mini\_train\_label==0)], train\_y[np.where(mini\_train\_label==0)], color='red')

plt.scatter(train\_x[np.where(mini\_train\_label==1)], train\_y[np.where(mini\_train\_label==1)], color='blue')

plt.scatter(train\_x[np.where(mini\_train\_label==2)], train\_y[np.where(mini\_train\_label==2)], color='yellow')

plt.scatter(train\_x[np.where(mini\_train\_label==3)], train\_y[np.where(mini\_train\_label==3)], color='black')

test\_x = mini\_test[:,0]

test\_y = mini\_test[:,1]

outputlabels = np.array(outputlabels)

plt.scatter(test\_x[np.where(outputlabels==0)], test\_y[np.where(outputlabels==0)], marker='^', color='red')

plt.scatter(test\_x[np.where(outputlabels==1)], test\_y[np.where(outputlabels==1)], marker='^', color='blue')

plt.scatter(test\_x[np.where(outputlabels==2)], test\_y[np.where(outputlabels==2)], marker='^', color='yellow')

plt.scatter(test\_x[np.where(outputlabels==3)], test\_y[np.where(outputlabels==3)], marker='^', color='black')

#save diagram as png file

plt.savefig("miniknn.png")

Program Procedure:

I use the template of the python code on sakai. But I import counter from collections library and then all my codes are in the defining KNN classifier area. I set an unreachable large number 999 to get the smallest L2 distance and then set it as 999. I use 2 for loop to calculate the L2 distance between the random generated sample point and the training data. Then I choose the K smallest data from those L2 distances and remember its index. I use the index to get the class of the closest training points. Finally I count the number of the most occurred class as the final result of where this sample point belongs to. The screen shots of output will be posted in **figure 2** and **figure 3**.

Output of Problem2:

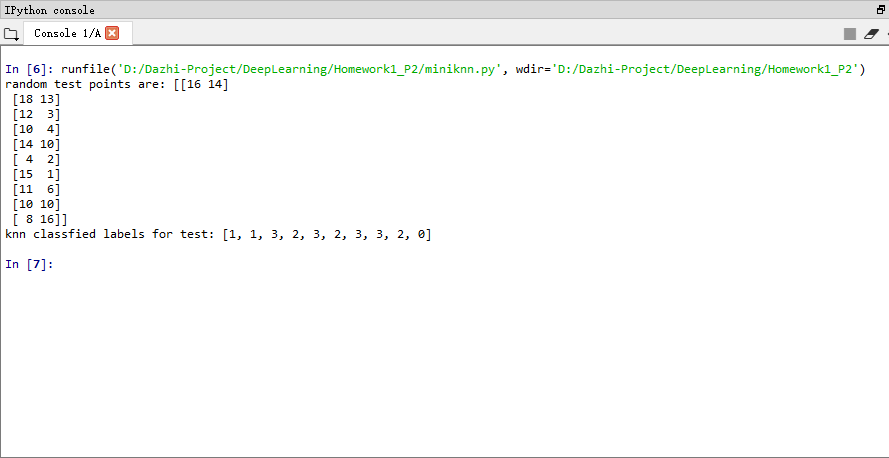


Figure 2

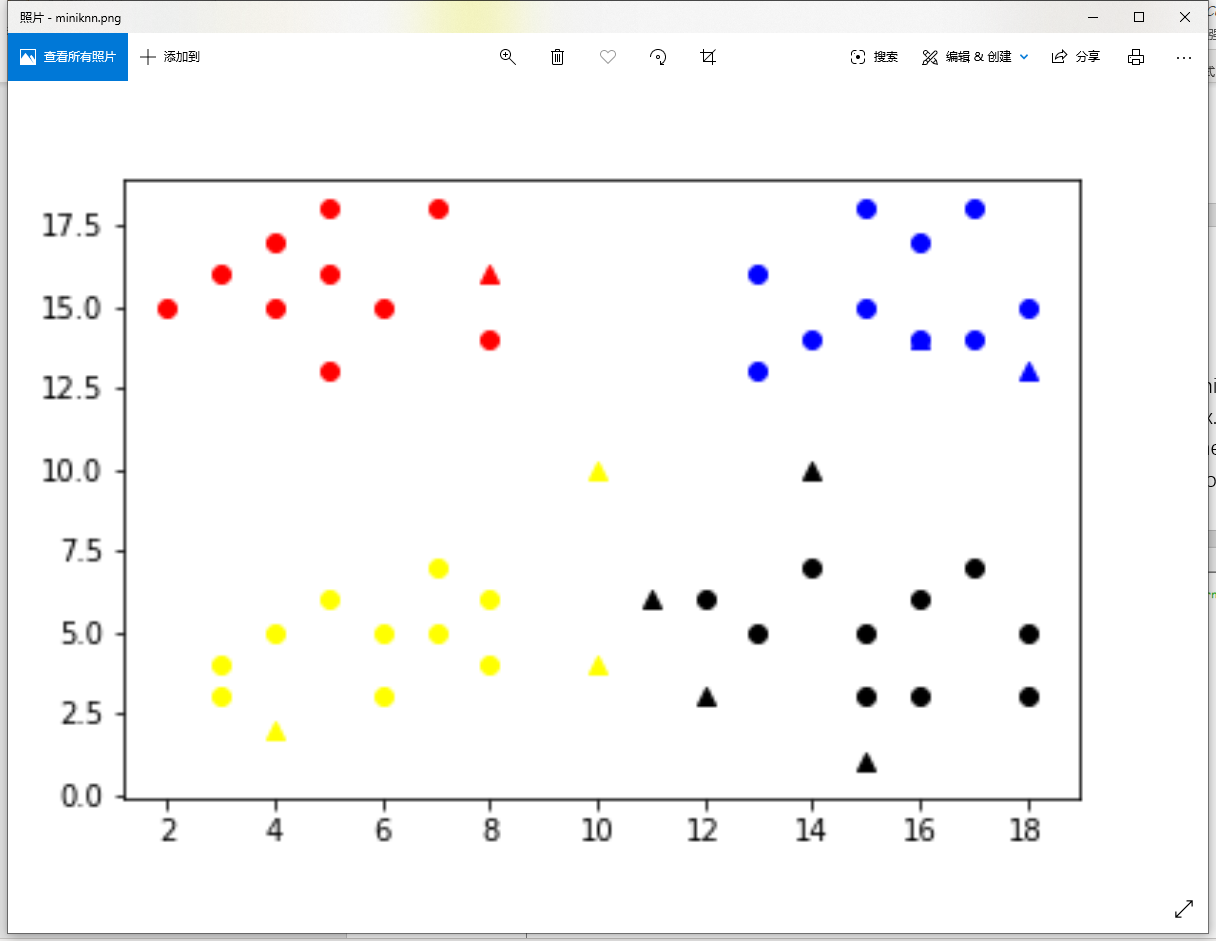


Figure 3

**Problem 3**

Source Code:

#import math

import numpy as np

from download\_mnist import load

#import operator

import time

from collections import Counter

# classify using kNN

#x\_train = np.load('../x\_train.npy')

#y\_train = np.load('../y\_train.npy')

#x\_test = np.load('../x\_test.npy')

#y\_test = np.load('../y\_test.npy')

x\_train, y\_train, x\_test, y\_test = load()

x\_train = x\_train.reshape(60000,28,28)

x\_test = x\_test.reshape(10000,28,28)

x\_train = x\_train.astype(float)

x\_test = x\_test.astype(float)

def kNNClassify(newInput, dataSet, labels, k):

result=[]

########################

# Input your code here #

########################

for i in newInput:

L2\_dist=0

L2=[]

for j in dataSet:

'''

This kind of method is low\_efficient and low\_accurate

for row in range(28):

for column in range(28):

L2\_dist+=np.sqrt((i[row][column]-j[row][column])\*\*2)

'''

L2\_dist=np.sum(np.sqrt((i-j)\*\*2))

L2.append(L2\_dist)

Min\_list=[]

for a in range(k):

Min\_list.append(L2.index(min(L2)))

L2[L2.index(min(L2))]=float("inf")

classifier=[]

for b in Min\_list:

classifier.append(labels[b])

#print(classifier)

result.append(Counter(classifier).most\_common(1)[0][0])

####################

# End of your code #

####################

return result

start\_time = time.time()

outputlabels=kNNClassify(x\_test[0:20],x\_train,y\_train,10)

result = y\_test[0:20] - outputlabels

result = (1 - np.count\_nonzero(result)/len(outputlabels))

print ("---classification accuracy for knn on mnist: %s ---" %result)

print ("---execution time: %s seconds ---" % (time.time() - start\_time))

Program Procedure:

I also use the template of python codes from the sakai. But I import some packages which is needed and interpret some unused packages. Since there are 60000 training data sets, I calculated there would be a lot of computation about 60000 \* 28 \* 28 \* number\_of\_testdata, which is a huge computation capacity. At first I chose the code as what it is in problem 2 but I found that the final output needed too much time about 1 min for a test data and accuracy was so low about 0.15. So I use the numpy library which could compute array very fast. The way to get K smallest L2 distances and its label is the same as what I do in problem 2. The final output will be shown in **figure 4**(20 test datasets) and **figure 5**(100 test datasets).

Output of Problem 3:

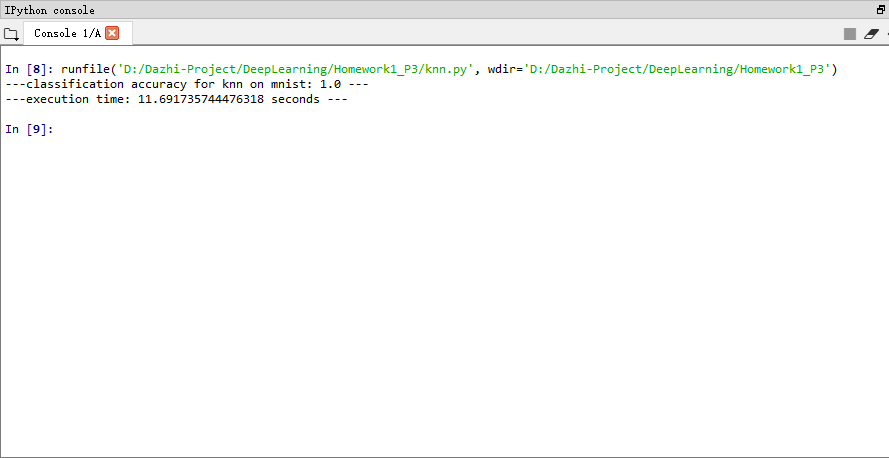


Figure 3-First 20 test datasets

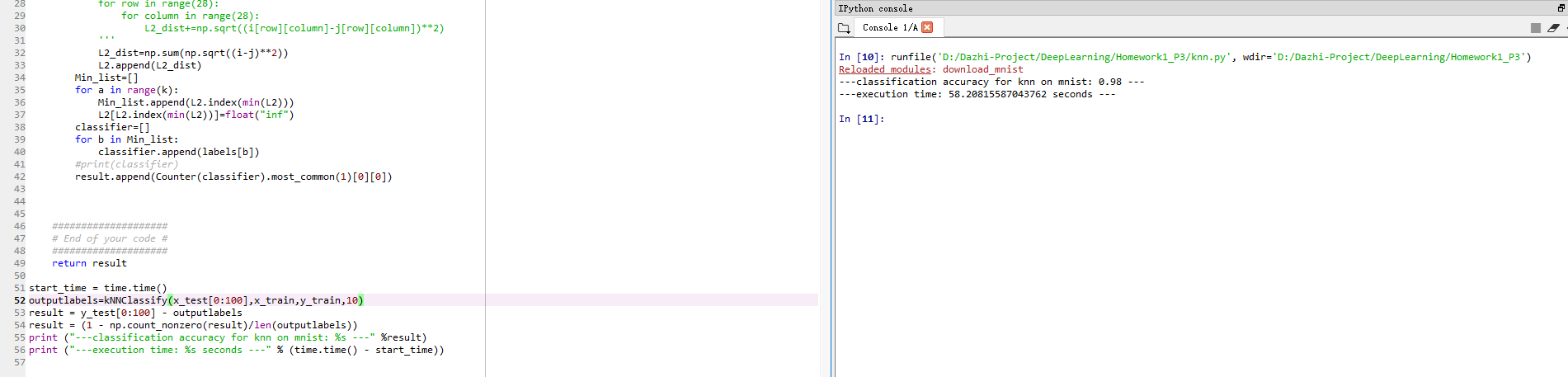


Figure 5-First 100 test datasets